CS/ECE 552: Introduction to Computer Architecture

Prof. David A. Wood

Final Exam
May 9, 2010
10:05am-12:05pm, 2241 Chamberlin
Approximate Weight: 25%

CLOSED BOOK
TWO SHEETS OF NOTES

NAME: _____

DO NOT OPEN THE EXAM UNTIL TOLD TO DO SO!

Read over the entire exam before beginning. Verify that your exam includes all 9 pages. It is a long exam, so use your time carefully. Budget your time according to the weight of the questions, and your ability to answer them. Limit your answers to the space provided, if possible. If not, write on the BACK OF THE SAME SHEET. Use the back of the sheet for scratch work. WRITE YOUR NAME ON EACH SHEET.

Problem	Possible Points	Points
Problem 1	20	
Problem 2	15	
Problem 3	20	
Problem 4	25	
Problem 5	20	
Total	**100**	

NAME: _____

Problem 1: (20 points)

Ideally, the 5-stage pipeline discussed in class (and the book) will complete one instruction every cycle. Stall Cycles Per Instruction (SCPI) is a metric that measures the average number of stalls (i.e., pipeline bubbles) that get introduced per instruction. Thus the processor's CPI = 1 + SCPI. SCPI can be expressed as the sum of the SCPI's of different (independent) factors. For example,

$$CPI = 1 + SCPI_{data} + SCPI_{branch} + SCPI_{I\text{-}cache} + SCPI_{D\text{-}cache}$$

breaks the CPI down into stalls due to data dependence stalls, branch stalls, instruction cache stalls, and data cache stalls.

Consider a processor and memory system with the following properties:

- Branches are predicted not-taken and resolved in Execute.
- Cache hits stall the pipeline one cycle; cache misses stall the pipeline 15 cycles.
- The only data hazards that stall the pipeline are caused by load-use dependences (1 cycle).

The workload has the following properties:

- Loads are 30% of instructions; stores are 15% of instructions
- Branches are 12.5% of instructions; 60% of branches are taken
- 33% of load instructions are immediately followed by a dependent instruction
- 3% of instruction fetches miss; 5% of load and store instructions miss

Part A: (12 points) Complete the table below. Show both the equation and the final value.

	Equation	Value
$SCPI_{data}$	**30% * 33% * 1 cycle**	**.099**
$SCPI_{branch}$	**12.5% * 60% * 2 cycles**	**.15**
$SCPI_{I\text{-}cache}$	**100% * (1 - 3%) hits * 1 + 3% * 15**	**1.42**
$SCPI_{D\text{-}cache}$	**(30% + 15%) * ((1 - 5%) * 1 + 5% * 15)**	**.765**
CPI	**$1 + SCPI_{data} + SCPI_{branch} + SCPI_{I\text{-}cache} + SCPI_{D\text{-}cache}$**	**3.43**

NAME: _____

Part B: (8 points)

Stalling on cache hits has a siginificant impact on performance. How much does the CPI change if we can eliminate stalls on cache hits? Complete the table and calculate the speedup:

	Equation	Value
$SCPI_{I\text{-}cache}$	**100% * 3% * 15**	**0.45**
$SCPI_{D\text{-}cache}$	**(30% + 15%) * 5% * 15**	**.3375**
CPI	**$1 + SCPI_{data} + SCPI_{branch} + SCPI_{I\text{-}cache} + SCPI_{D\text{-}cache}$**	**2.04**

$CPI_{old}/CPI_{new} = 3.43/2.04 = 1.68$

Speedup (assuming cycle time remains constant): __1.68__

NAME: _____

Problem 2: (15 points)

Part A: (3 points) Show the 1-bit Booth recoding for the 8-bit multiplier -99_{ten}.

$-99_{ten} = 10011101_{two} = -1\ 0\ 1\ 0\ 0\ -1\ 1\ -1$

Part B: (3 points) What is the 2-bit modified Booth recoding of the multiplier in Part A?

-2 2 -1 1

Part C: (4 points) A 16-bit multiplier has been recoded using the 2-bit modified Booth recoding algorithm. The recoded multiplier is:

```
-1  1  2  0  -1  -2  1  -1
```

What is the original 16-bit *two's complement* multiplier?

11 01 01 11 10 10 00 11

0-1 1-1 10 00 -11 -10 01 0-1

-1 1 2 0 -1 -2 1 -1

Part D: (5 points) How does recoding the multiplier using 2-bit modified Booth algorithm help improve the speed of the multiplication?

Recoding the multiplier with 1-bit Booth simplifies the treatment of negative numbers, but doesn't really help improve the speed of the multiplication. However, using the 2-bit algorithm reduces the number of partial products by half, which reduces the number of adders we need in our multiplier (e.g., the number of inputs to the Wallace tree).

NAME: _____

Problem 3: (20 points)

Consider a memory system with the following parameters. The virtual address has 56 bits. The physical address has 48 bits. A unified (i.e., instructions and data) cache is writeback and has 64 kilobyte capacity with 32-byte blocks, is 8-way set associative with pseudo-least-recently-used (pLRU) replacement (also called hierarchical nMRU). The translation lookaside buffer (TLB) has 96 entries, is fully associative, and is accessed *before* the cache. Pages are 4 kilobytes. All addresses are byte addresses.

Part A: (2 points) Show the breakup of a virtual address into virtual page number and byte offset within a page. Indicate the number of bits in each field.

Virtual Page Number<55:12> . Page Offset <11:0>

Part B: (2 points) Show the breakup of a physical address into a page frame number and a byte offset with the page frame. Indicate the number of bits in each field.

Page Frame Number<47:12> . PageOffset<11:0>

Part C: (3 points) How many bits of storage does it take to implement this TLB? Assume the *minimum* number of bits possible to achieve a correct implementation of address translation.

TLB entry = VPN + PFN + valid bit + page dirty = 44 + 36 + 1 = 81 bits/entry

96 entries * 82 bits/entry = 7,776

Part D: (3 points) What other bits of state may a TLB maintain? What are these used for?

Reference = Used by OS to approximate LRU or other page replacement policy

Dirty = Used by OS to reduce writeback traffic to disk

Protection = Use by OS to limit access to pages

Part E: (2 points) Show the break up of the physical address into tag, index, and byte within block used to access the cache. Indicate the number of bits in each field.

Tag<47:13> . Index <12:5> . BlockOffset<4:0>

Part F: (2 points) How many sets are there in the cache?

256 sets

Part G: (2 points) How many bits per set are needed to implement the pLRU replacement policy?

n-1 = 7

Part H: (4 points) How many total *bits* does it take to implement the cache (i.e., tags, state, data, LRU, etc.).

35 bit tag + 1 valid + 1 dirty + 8 * 32 data = 293/block

8 * 293 + 7 = 2351 bits per set

256 sets * 2351 bits per set = 601,856 bits

NAME: _____

Problem 4: (25 points)

Part A: (2 points) Odd parity is frequently used to ensure that all code words stored in memory have at least one 1. Consider a code word $pb_3b_2b_1b_0$ consisting of a parity bit p and four data bits $b_3b_2b_1b_0$. What is the equation to generate the parity bit p?

$p \wedge b_3 \wedge b_2 \wedge b_1 \wedge b_0 = 1$

$p = 1 \wedge b_3 \wedge b_2 \wedge b_1 \wedge b_0$

Part B: (2 points) What is the Hamming distance between $0011\ 0001_{two}$ and $1101\ 0111_{two}$?

5

Part C: (2 points) What is the minimum Hamming distance between any pair of valid code words encoded using odd parity?

2

Part D: (2 points) What is the minimum Hamming distance needed between any pair of valid code words to *correct* a single bit error?

3

Part E: (2 points) What is the minimum Hamming distance needed between any pair of valid code words to *detect* a single bit error?

2

Part F: (2 points) What is the minimum Hamming distance needed between any pair of valid code words to *correct* a single bit error AND *detect* a double bit error?

4

NAME: _____

Part G: (5 points) The parity check matrix for an error correcting code is given below. In this matrix C_i's denote check bits and b_i's denote information bits. The codewords are stored in memory in the bit order $C_3C_2C_1C_0b_3b_2b_1b_0$.

	b_3	b_2	b_1	b_0
C_0	1	1	0	1
C_1	1	0	1	1
C_2	0	1	1	0
C_3	1	1	1	1

Consider the data word $b_3b_2b_1b_0$ = 1010. Assuming *odd parity is used to compute all check bits*, what is the codeword that is stored in memory for the data word given the above parity check matrix?

1010 1010

Part H: (4 points) Suppose that the word read from the memory is 1100 0111, calculate the syndrome using the parity check matrix above.

Syndrome = c_3 ^ c_3', c_2 ^ c_2', c_1 ^ c_1', c_0 ^ c_0'

Syndrome = 1 ^ 0, 1 ^ 1, 0 ^ 1, 0 ^ 1 = 1011

Part I: (4 points) What is the procedure for detecting a double error?

If the syndrome is non-zero AND has an even number of bits set (i.e., even parity).

With two errors, the syndrome will either be the XOR of two names, two check bits, or a name and a check bit. Since all names have an odd number of bits, the syndome must have even parity.

Part J: (3 points, extra credit) Was there a double bit error in Part H? If not, what was the value originally written into memory? Like most real systems, ignore the possibility of more than two errors.

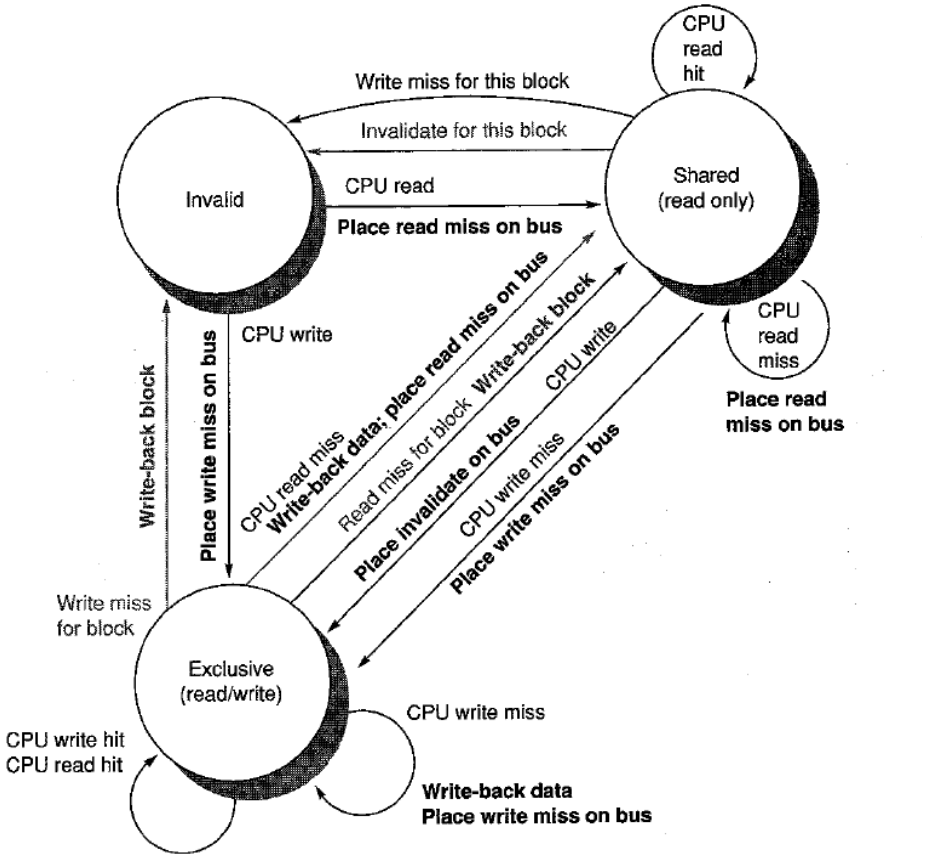
No, there was only a single bit error. The syndrome has an odd number of ones.

There is an error in the matrix, since there are two bits (b_3 and b_0) that have the same "name". We have no way of knowing which of these was the bit that flipped. I will give credit for either answer and two extra points for those that identify the problem.

Problem 5: (20 points)

Consider a bus-based multiprocessor system with writeback caches. Four processors, P1, P2, P3, and P4 perform the following sequence of loads and stores to/from lines A and B. Assume A and B do not conflict in the data caches. Assume the protocol on the next page, which uses the three states: Invalid (I), Shared (S), and Exclusive (E). The table below shows the state of the memory system as time flows down. The cache blocks are represented with the following notation: *address:(state, data)*. For example, *A:(I,0)* means that cache block A is in state I with data value 0. A data value of x means the value is unknown or undefined. Complete the table below, updating the cache and memory states in response to the sequence of loads and stores. Indicate actions taken by the cache and memory controllers: hits, requests to get a block shared or exclusive, and responses to requests. You may use arrows (as shown) to indicate that the state has not changed in that cycle.

P1	P2	P3	P4	MEM
A:(I,x) B:(I,x)	A:(I,x) B:(I,x)	A:(I,x) B:(I,x)	A:(I,x) B:(I,x)	A:0 B:0
load A miss, get A shared A:(S,0) B:(I,x)	↓	↓	↓	respond with A A:0 B:0
↓	**load B** miss, get B shared A:(I,x) B:(S,0)	↓	↓	respond with B A:0 B:0
↓	**load A** **miss, get A in S** **A:(S,0) B:(S,0)**	↓	↓	**respond with A** **A: 0 B:0**
↓	**store B = 1** **hit, send Inv** **A:(s,0) B:(E,1)**	↓	↓	**A: 0 B:0**
↓	↓	**load A** **miss, get A in S** **A:(S,0) B:(I,x)**	↓	**respond with A** **A: 0 B:0**
load B **miss Get B in S** **A:(S,0) B:(S,1)**	**respond with B** **A:(S,0) B:(S,1)**	↓	↓	**update B** **A:0 B:1**
invalidate A **A:(I,0) B:(S,1)**	**invalidate A** **A:(I,0) B:(S,1)**	**invalidate A** **A:(I,0) B:(I,x)**	**store A = 3** **miss, Get A in E** **A:(A,3) B:(I,x)**	**respond with A** **A: 0 B:1**
load A **miss, get A in S** **A:(S,3) B:(S,1)**	↓	↓	**respond with A** **A:(S,3) B:(I,x)**	**update A** **A:3 B:1**
↓	↓	↓	**load B** **miss, get B in S** **A:(S,3) B:(S,1)**	**Respond with B** **A:3 B: 1**


CPU
read
hit
Write miss for this block
Invalidate for this block
Invalid
CPU read
Place read miss on bus
Shared
(read only)
CPU write
Write-back block
Place write miss on bus
CPU read miss
Write-back data; place read miss on bus
Read miss for block
Write-back block
Place invalidate on bus
CPU write
CPU write miss
Place write miss on bus
CPU
read
miss
Place read
miss on bus
Write miss
for block
Exclusive
(read/write)
CPU write miss
CPU write hit
CPU read hit
Write-back data
Place write miss on bus